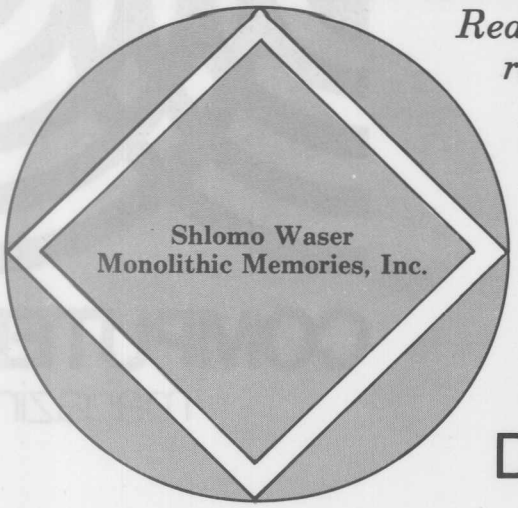
a reprint from

COMPUTER
magazine

HIGH SPEED MONOLITHIC MULTIPLIERS FOR REAL-TIME DIGITAL SIGNAL PROCESSING



Shlomo Waser
Monolithic Memories, Inc.

Real-time digital signal processing requires very fast multiplication, which is now becoming possible using mathematical techniques to take advantage of single-chip multipliers.

High-Speed Monolithic Multipliers for Real-Time Digital Signal Processing

In the past, most digital signal processing has required that digitized signals be recorded and then processed off-line on general-purpose computers. In many cases, however, on-line, real-time, and therefore very fast processing is required if digital techniques are to successfully replace analog techniques. The ultimate limitation on speed, and hence filter or spectrum-analyzer bandwidth, is multiplication speed. It has recently become feasible to implement fast multipliers on single silicon chips, and these monolithic devices promise to make real-time digital signal processing widely available.

In digital signal processing, a sampled point of the analyzed waveform may require one addition and one multiplication. The multiplication is traditionally performed by successive addition, so that for n-bit data words, n additions are required. For example, if the analog signal is quantized into eight bits, the required computation is made of nine additions—one for the real addition and eight simulating a multiplication. If the multiplication speed were to match the addition speed, the bandwidth of the signal processor would increase by a factor of four. For 16-bit words, a matching multiplication speed would improve the bandwidth by a factor of eight.

Simple multiplication: add and shift

The add-and-shift algorithm is the simplest way to perform a multiplication when only adding and shifting resources are available. The principle is similar to the way one multiplies numbers using pencil and paper. For example, multiplying two unsigned binary numbers is done as follows:

6	110
5	101
6×2^0	110
0×2^1	000
6×2^2	110
30	11110

When this operation is simulated in a software routine, each bit of the multiplier results in one add and one shift operation. Since most computers can add and shift in the same instruction cycle, at least n operations are required for n bits of multiplication. Figure 1 illustrates the MSI implementation of a sequential multiplier made up of an adder and a shift register. This sequential method is adequate for low-speed multiplication; for high-speed multiplication a combinatorial approach is needed.

In a combinatorial approach, the partial products are formed simultaneously and then added concurrently. Each bit in the partial product is formed by ANDing a multiplicand bit with a multiplier bit. This is related to the add-and-shift algorithm, since a multiplier bit of zero value results in zero partial product, and a multiplier bit of one value will make

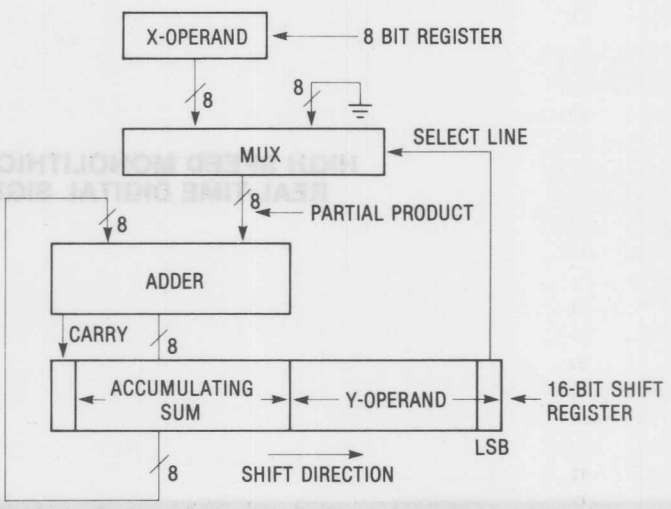


Figure 1. Eight-bit sequential multiplier made of MSI components. For simplicity, only the data paths are shown.

0018-9162/78/1000-0019$00.75 © 1978 IEEE

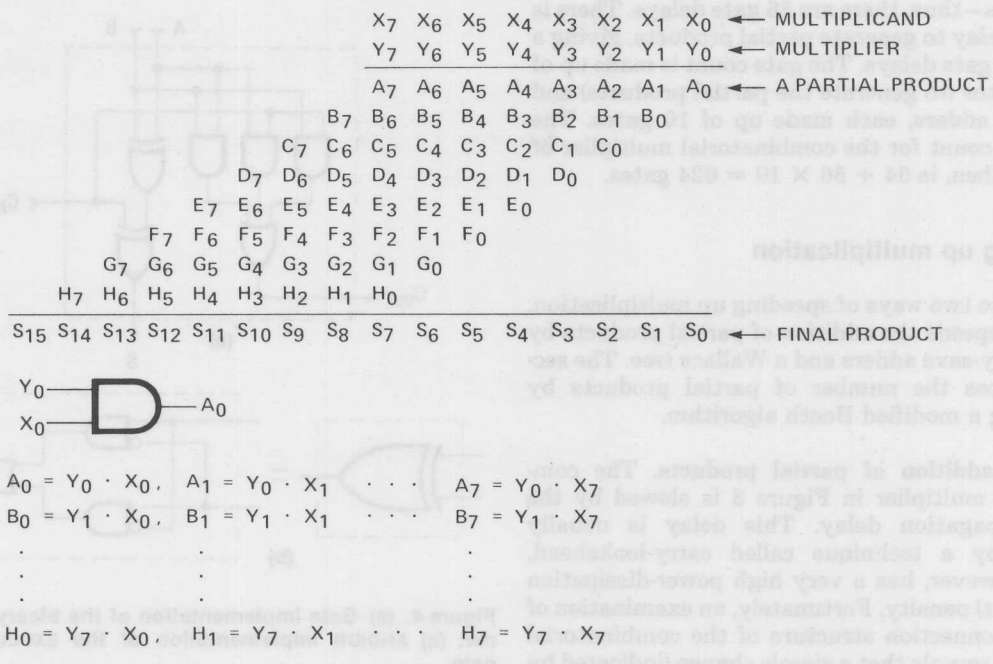


Figure 2. Generating and adding partial products in multiplication of two 8-bit operands.

the partial product equal to the multiplicand. Figure 2 illustrates the generation of partial products and their relative positions for the final double-length product. Figure 3 illustrates the interconnection of adders to sum the partial products.

To simplify the analysis of combinatorial multiplications, the speed of multiplication is expressed as (AND/OR) gate delays and the power dissipation as a gate count. Figure 4 shows the AND/OR gate implementation of the binary adder and the EXCLUSIVE-OR gate. In Figure 3, the speed of multiplication is governed by the carry-propagation delay—that is, the delays from A_1 to B_7 to S_{15}. This path is made up of 14 adder stages, each with four

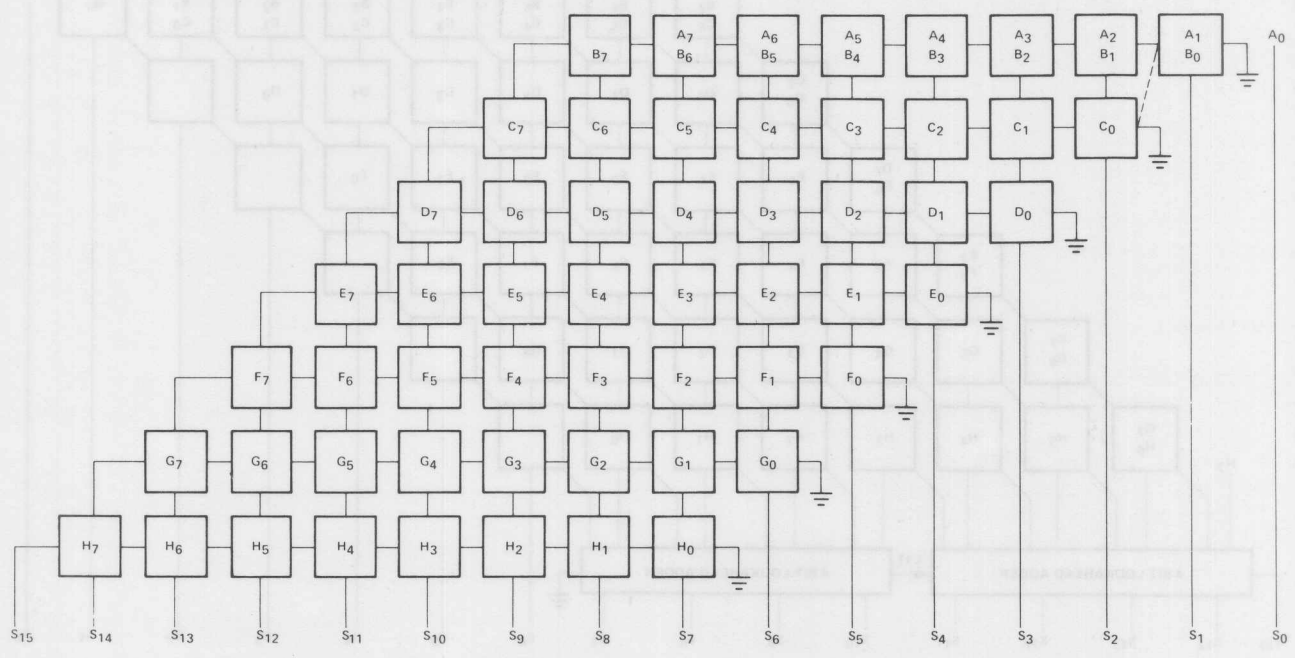


Figure 3. 8 x 8 combinatorial multiplier made up of 56 binary full adders. Multiplication speed is governed by the ripple carry.

gate delays—thus, there are 56 gate delays. There is one gate delay to generate partial products, giving a total of 57 gate delays. The gate count is made up of 64 AND gates (to generate the partial products) and 56 binary adders, each made up of 10 gates. The total gate count for the combinatorial multiplier of Figure 3, then, is $64 + 56 \times 10 = 624$ gates.

Speeding up multiplication

There are two ways of speeding up multiplication. The first speeds the addition of partial products by using carry-save adders and a Wallace tree. The second reduces the number of partial products by employing a modified Booth algorithm.

Faster addition of partial products. The combinatorial multiplier in Figure 3 is slowed by the carry-propagation delay. This delay is usually reduced by a technique called carry-lookahead, which, however, has a very high power-dissipation (gate count) penalty. Fortunately, an examination of the interconnection structure of the combinatorial multiplier reveals that a simple change (indicated by the dashed line) in the interconnections of carries can almost halve the carry-propagation delay. Instead of waiting for the carry to ripple, we add the carry at a later stage.

Postponement of the addition of carries can be extended to all adder stages except the last. (The carries from the last stage form an n-bit operand to be added to an n-bit sum, and this operation is done by carry lookahead.) Postponing addition of the carries to a later stage also permits a third input in each adder in the upper stage, so that the first three partial products can be added by the first stage, reducing the number of adder stages from seven to six. The adder used in this way has been called the carry-save adder.[1] This scheme is shown in Figure 5.

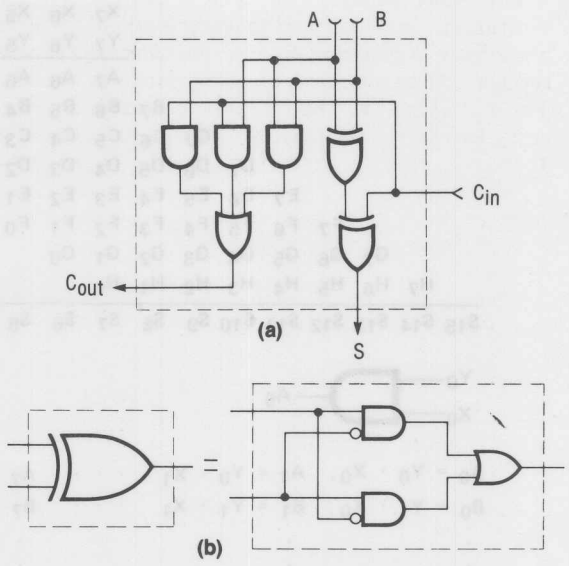


Figure 4. (a) Gate implementation of the binary full adder. (b) AND/OR implementation of the EXCLUSIVE-OR gate.

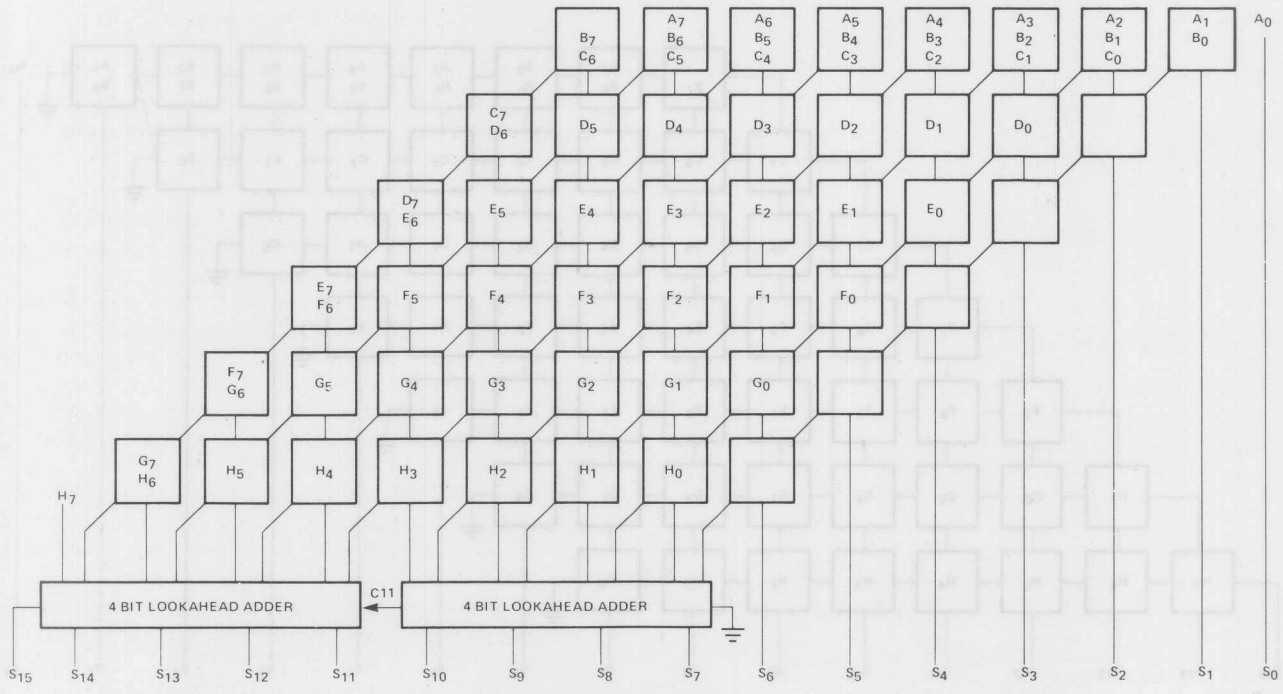


Figure 5. 8 x 8 combinatorial multiplier made of carry-save adders and carry-lookahead adders.

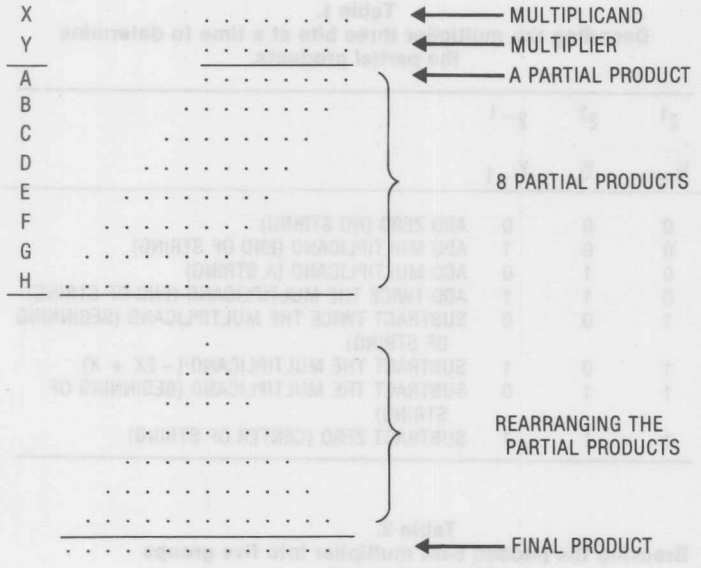


Figure 6. Dot representation of an 8 x 8 multiplication (compare to Figure 2).

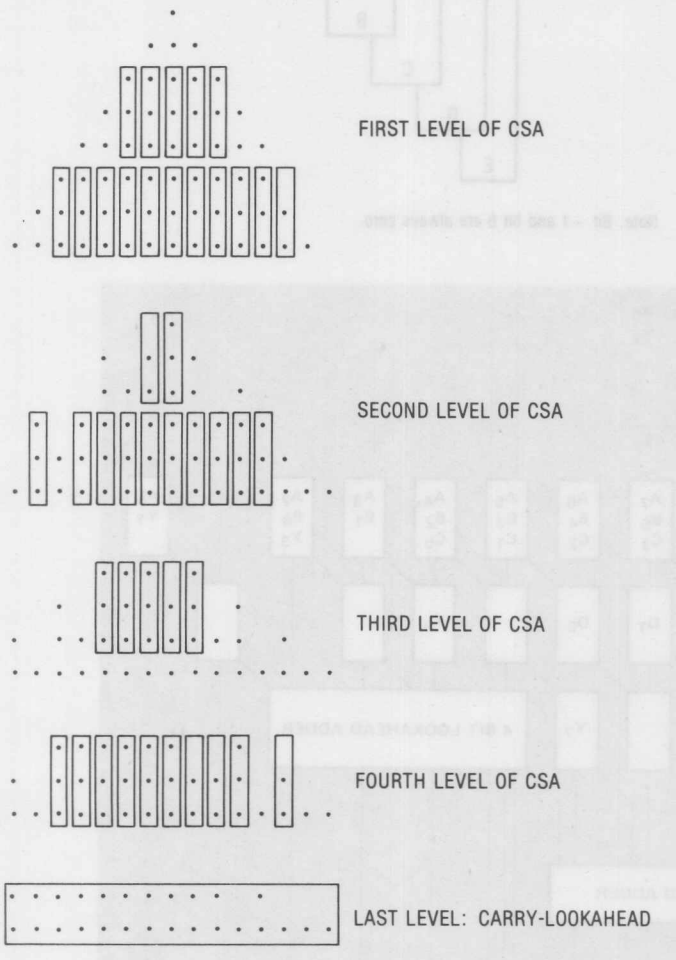


Figure 7. Wallace-tree reduction of 8 x 8 multiplication, using carry-save adders.

Additional speed can be gained by the use of a Wallace tree.[2] The Wallace tree is an interconnection of carry-save adders that reduces n partial products to two operands. The principle of the Wallace tree is to use a single carry-save adder to reduce three bits of equal weight to two bits, one a sum and the other a carry. In an $n \times n$ multiplication, n partial products are generated. These partial products can be viewed as adjacent columns of equal-weight bits. The maximum height of a column is n bits, and it can be divided into three-bit groups. Each of these groups can be reduced simultaneously to two bits, resulting in a new column $(2/3)n$ in height. The new column is again divided into three-bit groups, and the process repeats until the final column height is two bits.

To illustrate the Wallace-tree interconnection, we show a dot representation of an 8×8 multiplication in Figure 6. The Wallace-tree reduction network is depicted in Figure 7, where only four levels of carry-save adders are necessary to reduce the eight partial products to two operands (compared with the six levels of Figure 4). The propagation delay of the multiplier in Figure 7 is composed of one gate to generate partial products and four carry-save-adder delays (each four gates), and two stages of carry-lookahead (each three gates), resulting in 23 gate delays. The gate count is 64 + 44 (CSA) + 2 (CLA) = 110. Each carry-lookahead unit is made up of 30 gates, resulting in a total gate count of $64 + 44 \times 10 + 2 \times 30 = 564$ gates. Thus, the Wallace tree reduces the gate count and increases the speed of the multiplication.

In general, for a Wallace tree, the number of carry-save-adder delays necessary to reduce n partial products to two operands is:

$$\text{CSA delays} = \lceil \log_{3/2} n \rceil - 1$$

Thus, in a 32×32-bit multiplication, 8 CSA delays are needed to reduce the 32 partial products to two operands, compared to 30 CSA delays using the simple interconnection of Figure 4.

Fewer partial products. Another speed-up technique, a modification of Booth's algorithm,[3] increases speed by reducing the number of partial products by a factor of two; this reduces the number of CSA stages and, of course, the gate count. The purpose of Booth algorithms is to skip over a string of bits rather than form a partial product for each bit. Skipping the 0's in a string is quite clear; skipping the 1's uses a special property of the string. A string of 1's can be computed by subtracting the weight of the rightmost 1 from the modulus of the string; for example, the binary string 1111 is computed as $2^4 - 2^0 = 15$, and the binary string 11100 is computed as $2^5 - 2^2 = 28$.

In the hardware implementation of the modified Booth algorithm, the operand (multiplier) is divided into substrings of three bits each, and all possible permutations are computed from Table 1. Booth's algorithm (for unsigned numbers) requires that the

operand be padded with a zero to the left of the MSB, and a zero to the right of the LSB; thus, an eight-bit operand is divided into five groups, each of three bits. Every two contiguous groups have one bit in common (see Table 2). Thus, the modified Booth algorithm[1] is a multiplier-encoding scheme that involves constant shifting of 2 bits at a time while examining three multiplier bits, resulting in five partial products rather than the eight without encoding.

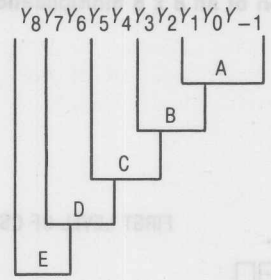
Like all good things, the modified Booth algorithm has a catch: a subtraction is now required. The subtraction is implemented by adding the two's complement of the number to be subtracted. Taking the two's complement of a number involves first complementing the number then adding 1 to its LSB. Table 1 shows that the bit Y_{i+1} can be used to indicate subtraction; thus the value of Y_{i+1} can be added to the LSB of the partial product. If $Y_{i+1} = 0$, no subtraction is called for and adding 0 changes nothing; on the other hand, if $Y_{i+1} = 1$, then the proper two's complement is performed by adding 1 to the LSB. In the two's complement, the sign bit must be extended to the full width of the final result. This is shown in Figure 8(a) as A_9 for the first partial product, B_9 for the second, etc. A proof of the modified Booth algorithm was given by Rubinfield.[4]

The gate delay of Figure 8(b) consists of two gates for decoding Table 1, two gates for selecting x or 2x, two CSA stages, and three stages of carry-lookahead, giving a total of 21 gate delays. The gate count includes five encoders, 32 multiplexers, 40 carry-save adders, and three carry-lookahead ad-

Table 1.
Decoding the multiplier three bits at a time to determine the partial products.

2^1	2^2	2^{-1}	
Y_{i+1}	Y_i	Y_{i-1}	
0	0	0	ADD ZERO (NO STRING)
0	0	1	ADD MULTIPLICAND (END OF STRING)
0	1	0	ADD MULTIPLICAND (A STRING)
0	1	1	ADD TWICE THE MULTIPLICAND (END OF STRING)
1	0	0	SUBTRACT TWICE THE MULTIPLICAND (BEGINNING OF STRING)
1	0	1	SUBTRACT THE MULTIPLICAND ($-2X + X$)
1	1	0	SUBTRACT THE MULTIPLICAND (BEGINNING OF STRING)
1	1	1	SUBTRACT ZERO (CENTER OF STRING)

Table 2.
Breaking the padded 8-bit multiplier into five groups of three bits each.



Note: Bit −1 and bit 8 are always zero.

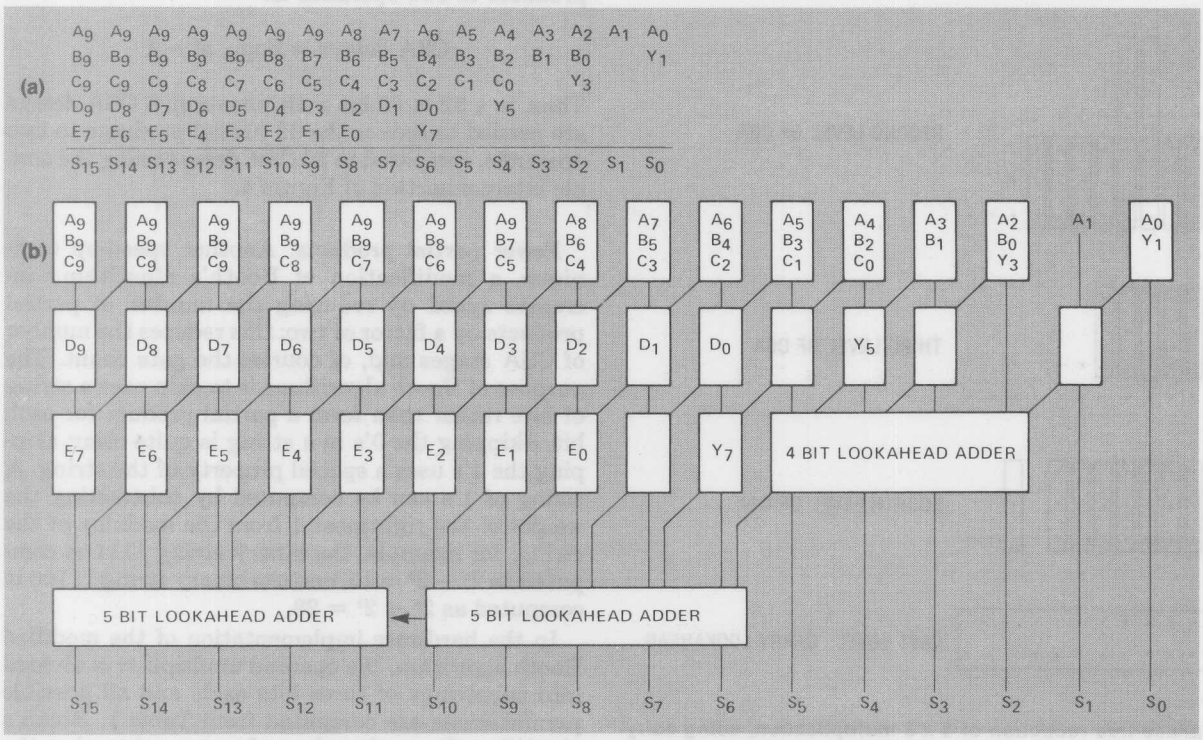


Figure 8. Array of adders for the 8 x 8 combinatorial multiplier.

ders, resulting in a total of $5 \times 5 + 32 \times 5 + 40 \times 10 + 3 \times 30 = 675$ gates. Thus, for a small penalty in gate count, the speed is improved, and the data is representable in either signed or unsigned notation.[5] Table 3 compares the gate count and the gate delay for all the combinatorial multiplier schemes.

The modified Booth algorithm encodes three bits at a time but could be extended to encode four bits at a time, which would reduce the number of partial products from $n/2$ to $n/3$. However, encoding four bits requires generating three times the multiplicand, which is not as trivial as generating two times the multiplicand.

Bounds on multiplication speed

Winograd derived the theoretical lower bounds (fastest times) for addition[6] and multiplication.[7] (The results are presented in an informal manner by Brennan.[8]) Winograd derived the bounds in terms of an (r,d) circuit, which is a d-valued logical circuit in which each element has fan-in at most r and can compute any r-argument d-valued logical function in unit time. The lower bound of addition of two operands, each made up of n bits, is proven to be (in the binary number system):

$$t \geq \lceil \log_r 2n \rceil$$

Winograd showed that the theoretical lower bound on multiplication speed is the same or slightly faster than the theoretical lower bound on addition speed: $t \geq \lceil \log_r \cdot (n-2) \rceil$. It is interesting to compare the speeds of practical circuits to the theoretical lower bounds. The most common scheme for implementing addition is carry-lookahead, and its speed is $4\lceil \log_r n \rceil$. In multiplication, the fastest practical realization uses multiplier encoding and a Wallace-tree interconnection of carry-save adders, resulting in a speed of $2\lceil \log_{3/2}(n) \rceil + 2 \lceil \log_r n \rceil$. To illustrate these results by numbers, assume $r=4$ (typical on chip fan-in) and $n=16$ bits. The lower bound for addition and multiplication is then three gate delays, whereas the carry-lookahead addition speed is eight gate delays and the multiplier-encoding/Wallace-tree multiplication speed is 18 gate delays.

Why is practical addition much closer to the lower bound than practical multiplication? The answer is that the conventional binary data representation is idealized for addition and not for multiplication. Brennan[8] states a principle explained by Winograd: any data representation permitting the lower bound of one operation to be approached cannot accomplish the same for the other. This principle is illustrated by the slide rule, which can perform multiplications (using logarithms) but not additions. The principle is also illustrated by using a ROM as a table lookup. The ROM can be thought of as an inefficient data representation for both addition and multiplication, and as such it does provide the same speed (access time) for both operations.

Table 3.
Comparison of the three combinatorial multiplier schemes.

	GATE DELAY	GATE COUNT	DATA REPRESENTATION
FIGURE 3	57	624	UNSIGNED ONLY
FIGURE 7	23	564	SIGNED ONLY
FIGURE 8	21	675	SIGNED AND UNSIGNED

Arithmetic requirements for signal processing

Speed. A rule states that in order to digitally recover a waveform, the sampling rate has to be at least twice the highest frequency component in the analyzed waveform. In speech processing, for example, where the bandwidth of interest is about 4 kHz, a sampling rate of 8 kHz, or a sample every 125 microseconds, is required.

Having determined the sampling rate, we must then ask how fast each computation must be. The answer depends on what type of analysis is to be performed. If, for example, the above speech waveform is analyzed by autocorrelation over a range of 256 samples, then each sample value is multiplied by the preceding 255 samples. In a period of 125 microseconds, 255 multiplications are required; therefore, each multiplication should not take more than 500 nanoseconds.

Word length. The question of word length (number of bits) comes up twice: at the time of input and during computation. The analog input signal is converted to a digital word. If the signal is weak and the interference relatively strong, the interference may be clipped along with the signal. When the clipped interference is analyzed, it will introduce spurious frequency components. For example, if the signal is 160 millivolts and each quantizing level is 10 millivolts, then four bits are sufficient for the A-D conversion. If the system uses eight bits, then a maximum interference of $2^8 \times 10$ mv $= 2560$ mv can be tolerated.

Number of bits is also of interest during computation. Textbook formulas on discrete time analysis always assume infinite-length data words, which are approximated to a greater or lesser extent by floating-point arithmetic in scientific computers. In real-time hardware implementations, the question of how closely to approximate must be analyzed carefully. Too many bits will result in expensive hardware, whereas too few bits will result in erroneous analysis.

The number of bits can also be an issue when the result of a computation requires more bits than the operands. The product of two operands each eight bits long, is 16 bits long. If this product needs to be multiplied again, much computation would be saved if it were truncated back to eight bits. But this cannot always be done, since the accumulated truncation errors may distort the final result. A com-

promise is to round rather than truncate. Rounding is accomplished by adding the weight of the MSB of the discarded part to the double-length product, which, after truncation, will result in a single-length rounded product.

Characterizing monolithic multipliers

Speed, power dissipation, and word length. The three most critical characteristics of a multiplier are speed of multiplication, power dissipation, and word length (number of bits). We have discussed above the selection of speed and number of bits for the overall computation. These results typically do not map directly into a monolithic multiplier. For example, the number of bits desired needs to be mapped into the existing common word lengths of 4, 8, 12, and 16 bits. The word lengths refer to the number of bits in each operand, and to half the number of bits in the final product. The range of speed in monolithic multipliers is from 40 nanoseconds on a 2 × 4 multiplier (the Advanced Micro Devices 25S05) to 800 nanoseconds on a 16 × 16 multiplier (the Monolithic Memories, Inc. 67516). If the number of bits required exceeds 16, then several multipliers are cascaded together to form the longer word; expansion techniques are described later.

The power dissipation of monolithic multipliers ranges from 300 milliwatts on a 2 × 4 multiplier (the AMD 25L05[9]) to 5 watts on a 16 × 16 multiplier (the TRW-MPY16[10]). The 5-watt dissipation requires special cooling, which brings up the question, at what point is special cooling required? To answer this, a thermodynamic relationship is called for: T junction = T ambient + $\Theta jA \times$ power, where T junction is the temperature of the silicon chip, T ambient is the still-air ambient temperature, ΘjA is the thermal resistance of the package, and the last term is the power dissipation. The thermal resistance is expressed as Δ °C per watt; in other words, one watt of heat energy will raise the temperature by Δ °C. For a 40-pin ceramic package, this value is 30 °C per watt. If the ambient temperature may be 125 °C (the standard military specification) and the maximum allowed silicon junction temperature is about 175 °C, then the maximum power dissipation that does not require special cooling is:

$$\text{max power} = (T \text{ junction} - T \text{ ambient})/\Theta jA$$
$$\text{max power} = (\ 175\ -\ 125\ )/30 = 1.6 \text{ watt}$$

These numbers are quite typical, so one watt is approximately the maximum power dissipation an LSI device can tolerate. In the case of the TRW 5-watt device, the package has cooling fins on it, which reduces the thermal resistance to 10 °C per watt. In general, TRW is the exception; most other LSI manufacturers are not exceeding the 1.6 watt power dissipation per single chip.

In addition to the criteria of speed, word length, and power dissipation, there is a figure of merit that allows comparison of multipliers with different word lengths. We call it normalized speed-power product. It is expressed as f = speed × power(number of bits)^2, and is justified informally by examining the number of gates and adders used in implementing a simple multiplier (Figure 3). It is further supported by empirical results. For example, TRW introduced at about the same time three multipliers whose figures of merit are about the same, as can be expected given the same technology maturity.

Other features. Some features of secondary importance should also be considered. They are expandability, data representation, and rounding.

Expandability refers to the building of a large multiplier array from several monolithic multipliers. One can easily judge the expandability of a multiplier unit by inspecting the number of additional non-multiplier chips that are needed to expand to larger multiplications. However, we will see that expandability comes at the expense of multiplication speed.

Data representation of the multipliers is either unsigned or signed (two's complement). Signed is more popular and is more frequently implemented, but it requires more hardware to accommodate sign extension and complementing during expansion. The MMI[11] 8 × 8 multiplier can accommodate both signed and unsigned representations.

Chip rounding is very useful when a single-length product is desired. Rounding is accomplished by raising the rounding pin to logic one, which propagates a carry into the least significant bit (of the single-length product) if the most significant bit of the lower half is one.

Parallel multipliers. There are two schemes for parallel multiplication[12]—iterative arrays of cells (AMD 25S05) and generation of a matrix of partial product terms (MMI 67558) with subsequent reduction of the matrix by means of psuedo adders. The array scheme uses only one IC type to implement any size of multiplication and is a relatively compact solution, but the speed increases linearly with the length of the operand. Matrix-generation schemes are much faster for larger operands, since here the speed of operation increases with the log of the operand length.

The AMD 25S05 is an iterative cell that implements $X \cdot Y + K$, where K is an input fed by the partial product from an earlier stage. In the early and mid 70's, this MSI device (which performs only 2 × 4 multiplication) was the standard building block for high-speed digital signal processing where the resultant large number of chips was not an objection. The device itself uses the modified Booth algorithm (in combinatorial fashion). A single 25S05 performs 2 × 4 multiplication at 25 nanoseconds, but 16 × 16 multiplication requires 32 devices and the associated delay is 150 nanoseconds. Motorola has a similar device (10183) implemented in ECL that performs 16 × 16 multiplication in 100 nanoseconds.

The MMI 67558 is an 8 × 8 matrix-reduction multiplier. Internally, it uses the modified Booth algorithm to generate five partial products that are reduced to two operands using carry-save adders. The resultant two operands are added using the carry-lookahead scheme. The device is housed in a 40-pin package, and its delay is 100 nanoseconds. The input data is interpreted (according to a mode-control line) as either signed two's complement or unsigned. The dual data representation simplifies expansion of the multiplier to a larger signed multiplication. For example, in multiplying 16-bit operands using 8-bit multipliers, the operand is broken into the most significant signed portion and the least significant unsigned portion.

TRW manufactures a series of non-expandable multipliers that are intended specifically for digital signal processing. In such applications, pipelining is an acceptable solution to obtain more bandwidth. The TRW multipliers, therefore, have input and output registers that can be clocked simultaneously. TRW solved the non-expandability problem by offering a series of multipliers at 8, 12, 16 and 24 bits. These multipliers are capable of operating at a pipeline rate of 100-200 nanoseconds. TRW recently extended its multiplier family to also perform accumulation. The accumulator has 27 bits, while only 24 bits are required for a double-length product. The extra bits enable accumulation of at least eight products before overflow occurs. The multiplier/accumulator that performs sum of products is an important building block for digital filters and fast-Fourier-transform systems.

The internal logic design of the TRW multiplier array is straightforward. Partial products are generated by AND gates and are then added using carry-save adders. This does not provide the greatest possible speed, but it has the important advantage of being very modular, which shortens the LSI design cycle and cuts initial design costs.

TRW had to solve two problems with the 16 × 16 multiplier: pin limitation and power dissipation. The 16 × 16 multiplier requires more than 64 pins. Because TRW wanted to use the standard 64-pin package, they had to multiplex the least significant portion of the product with one of the operands. This is quite acceptable, since a single-length product is often sufficient in signal processing. The excessive power dissipation (5 watts) was solved by mounting a heat sink on the package.

Serial-parallel multipliers. Serial-parallel multipliers multiply by sequentially computing and accumulating partial products. Although they are

Table 4.
Comparison of the available multiplier IC's.

DEVICE	CONFIG-URATION	PKG PINS	MAX DELAY (ns)	MAX POWER (WATTS)	NORMALIZED SPEED-POWER PRODUCT	DATA CODE	AMOUNT OF PARALLELISM	ALGORITHM
TRW MPY-8	8 × 8	40	170	1.5	4.0	TWO'S COMPLEMENT	FULL	AND GATES AND CARRY-SAVE-ADDERS
TRW MPY-12	12 × 12	64	200	3.75	5.2			
TRW MPY-16	16 × 16	64	230	5	4.5			
MMI 67558	8 × 8	40	125	1.4	2.7	2'S COMP AND UNSIGNED	FULL	MODIFIED BOOTH'S AND CARRY-SAVE-ADDERS
MMI 67516	16 × 16	24	800	1.3	4.1	2'S COMP	MULTIPLICAND AND 2 MULTIPLIER BITS	MODIFIED BOOTH'S
MMI 67508	8 × 8	20	400	0.75	4.7			
AMD 25S05	2 × 4	24	40	0.9	4.5	2'S COMP	FULL	BOOTH'S (ITERATIVE ARRAY)
AMD 25LS14	8 × 1	16	50	0.6	3.7	2'S COMP	MULTIPLICAND AND 1 MULTIPLIER BIT	BOOTH'S
TI 74S274 (ROM)	4 × 4	20	70	0.6	2.6	UNSIGNED	FULL	ROM LOOK-UP TABLE
MOT 10183	2 × 4	24	25	1.0	2.5	2'S COMP	FULL	BOOTH'S (ITERATIVE ARRAY)
AMD 25LS2516	8 × 8	40	400	1.4	8.7	2'S COMP	LIKE MMI67516	MODIFIED BOOTH'S

slower than parallel multipliers, more integration is possible for the same silicon area. For example, the 100-nanosecond 8 × 8 multiplier has about the same chip size as a 1-microsecond 16 × 16 multiplier/divider.

The 25LS14 from AMD was the first available serial-parallel multiplier. This 8 × 1 multiplier has one serial input and eight parallel inputs. Its most attractive feature is its easy expandability. In general, to do $n \times n$ multiplication requires only $n/8$ multipliers, but the number of clock cycles is $2n$. Thus, if larger multiplication is required and speed is not critical, this expandable approach is the best available. The author was told about an encryption application that requires 400 × 400 multiplication; using 50 25LS14's, it was accomplished in 32 microseconds. Another application that lends itself to a serial-parallel multiplier is single-line communication, where incoming information has to be passed through a digital filter. In this case, the serial data is fed into the serial port, and the known coefficient is loaded into the parallel port.

The AMD 25LS2516 is an 8 × 8 expandable multiplier that requires a 40-pin package to accommodate all the expansion interconnection. Its structure is an integration of three chips: an 8 × 1 serial multiplier, a parallel-in/serial-out shift register, and a serial adder. Its overall performance is similar to that obtained with the 25LS14.

The MMI 67516 is a non-expandable 16 × 16 multiplier/divider.[13] (A similar 8-bit version is available as the 67508.) Expandability was sacrificed for the sake of a smaller (24-pin) package and more capabilities (such as division). The logic design can be internally expanded to 24 and even 32 bits, but the present limitation is power dissipation. The 67516 uses the modified Booth algorithm to perform 16-bit multiplication in eight clock cycles (100 nanoseconds each). It can execute 23 different microinstructions; examples are multiply by a constant, multiply and accumulate, and divide. Division is accomplished using a non-restoring algorithm that takes 20 clock cycles (2 microseconds total).

The development of the 67516 was originally motivated by speech processing requirements, where 16-bit multiplications in one microsecond are used to implement digital filter equations. Later, it was realized that most of the data paths and registers needed for division already existed, so with a small increase in hardware and expansion of the microprogram, division capability was incorporated into the chip.

Table 4 lists the available monolithic multipliers and their main characteristics. Table 5 shows the performance of both single-chip and multiple-chip realizations of 8 × 8 and 16 × 16 multipliers. These two word lengths are capable of satisfying many digital signal-processing applications. Eight bits are commonly used in digital telephones (PCM), spectrum analysis, medical electronics, etc. Sixteen bits are common in speech processing, music synthesis, and image processing.

Table 5.
Performance comparison of 8 x 8 and 16 x 16 multiplications.

MULTIPLIER CHIP			8 × 8 MULTIPLICATION			16 ×16 MULTIPLICATION		
DEVICE	CONFIG-URATION	PKG PINS	NO. OF PKGS	MAX DELAY (ns)	MAX POWER (WATTS)	NO. OF PKG	MAX DELAY (ns)	MAX POWER (WATTS)
MMI 67558	8 × 8	40	1	125	1.4	(NOTE 1) 14	170	10
TRW MPY-8	8 × 8	40	1	170	1.5	(NOTE 1) 14	210	10
MOT 10183	2 × 4	24	8	75	8	32	150	32
AMD 25S05	2 × 4	24	8	100	7	32	220	27
TI 74S274	4 × 4	20	(NOTE 2) 12	100	7	45	150	27
TRW MPY-16	16 × 16	64	—	—	—	1	230	5
MMI 67516	16 × 16	24	—	—	—	1	800	1.3
AMD 25LS2516	8 × 8	40	1	400	1	2	800	2

Note 1. Four packages are 8 x 8 multipliers, 10 are adders
Note 2. Four packages are 4 x 4 multipliers, 8 more are Wallace tree bit slices (74S275)

Multiplication using ROMs

There are two ways in which ROM (PROM, EPROM, NVM, etc.) can be used to implement multiplication: by direct lookup and by using logarithms.

Direct lookup. For a ROM to be used as a lookup table for multiplying $m \times n$ bits, 2^{m+n} addresses and $m + n$ outputs are required for a precise product. The TI 74S274[14] is a 4 × 4 multiplier implemented in a 256 × 8 ROM. To implement 8 × 8 multiplication in ROM would require 64K × 16 ROM, whereas the largest existing (bipolar-technology) chip is only 2K × 8. However, since a certain amount of error is already present in a digitized signal because of quantization, rounding can be used. If the product is rounded to a single length, the percentage error is $0.5 \times 100\%/(2^n-1)$. If $n = 8$, the error is 0.2%, which is quite acceptable for many digital signal-processing applications, but the ROM size is still excessive (64K × 8 ≈ one half million bits).

Use of logarithms. In their excellent paper,[15] Brubaker and Becker show that, for a given acceptable error, it is possible to reduce considerably the number of ROM bits required for multiplication. Their discussion is based on multiplication using logarithms, which is described by the following relationship: $XY = \text{antilog}(\log X + \log Y)$. By expressing the multiplicand and multiplier in a type of floating-point format, they were able to reduce the number of bits by more than an order of magnitude. For example, for 0.5% error in 8 × 8 multiplication, only 6144 bits are required compared to 114,688 using direct multiplication lengths. But the penalty in using the logarithmic approach is in multiplication speed. Direct multiplication is delayed by the time of one ROM access, whereas logarithmic multiplication requires two ROM accesses plus an addition.

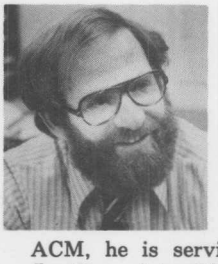
Future trends

These trends can be evaluated in terms of speed, resolution, and architecture. Multiplication speeds of 100 nanoseconds are quite adequate for many real-time digital signal-processing applications, but the 8-16-bit resolution available at this speed is not always sufficient, so it can be expected that fast 24 × 24 monolithic multipliers will become available. Further increases in resolution and range will be achieved by floating-point multipliers.

For medium-speed applications (0.5-1 microsecond) the architectures of the monolithic multipliers will be expanded to become programmable arithmetic processors. The trend in this direction is exemplified by two recent devices. One is the MMI 67516, which performs division and sum of products in addition to multiplication. The second is the Signal Processing Arithmetic Unit from TRW, which does parallel addition, subtraction, multiplication, scaling, and parallel-register multiplexing.[16] ■

References

1. O. L. MacSorley, "High-Speed Arithmetic in Binary Computers," *Proc. IRE*, Vol. 49, Jan. 1961, pp. 67-91.
2. C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. Electronic Computers*, Vol. EC-13, Feb. 1964, pp. 14-17.
3. A. D. Booth, "A Signed Binary Multiplication Technique," *Quart. J. Mech. Appl. Math.*, Vol. 4, Part 2, 1951.
4. L. P. Rubinfield, "A Proof of the Modified Booth's Algorithm for Multiplication," *IEEE Trans. Computers*, Vol. C-24, Oct. 1975, pp. 1014-1015.
5. S. Waser and A. Peterson, "Real Time Processing Gains Ground with Fast Digital Multiplier," *Electronics*, Sept. 29, 1977.
6. S. Winegrad, "On the Time Required to Perform Addition," *JACM*, Vol. 12, No. 2, 1965, pp. 277-285.
7. _____, "On the Time Required to Perform Multiplication," *JACM*, Vol. 14, No. 4, 1967, pp. 793-802.
8. J. F. Brennan, "The Fastest Time of Addition and Multiplication," *IBM Research Reports*, Vol. 4, No. 1, 1968.
9. Advanced Micro Devices, Inc., *Low Power Schottky Data Book*, Sunnyvale, Calif., 1977.
10. TRW, *MPY-Series Multipliers*, Redondo Beach, Calif., Oct. 1977.
11. Monolithic Memories, Inc., *67558 Data Sheet*, Sunnyvale, Calif., Aug. 1978.
12. W. J. Stenzel, et al., "A Compact High Speed Multiplication Scheme," *IEEE Trans. Computers*, Vol. C-26, No. 10, Oct. 1977.
13. Monolithic Memories, Inc., *67516/67508 Data Sheet*, Sunnyvale, Calif., Oct. 1977.
14. Texas Instruments, Inc., *The TTL Data Book for Design Engineers*, Dallas, Tex., 1976.
15. T. A. Brubaker and J. C. Becker, "Multiplication Using Logarithms Implemented with Read Only Memories," *IEEE Trans. Computers*, Vol. C-24, Aug. 1975, pp. 761-765.
16. J. L. Buie and T. A. Zimmerman, "Very Large Scale Integrated Circuits for Digital Signal Processing," *Circuits and Systems*, Feb. 1977, pp. 2-8.



Shlomo Waser is a product planning manager at Monolithic Memories, Inc., where he is responsible for definition and application support of logic and arithmetic integrated circuits. His background includes computer and test-equipment design. Before joining MMI in 1972, he was a designer at Information Storage Systems.

A member of the IEEE and the ACM, he is serving on the standards committee for floating-point arithmetic. He holds the BSEE and MSEE from San Jose State University, and is completing requirements for the Engineer degree from Stanford University.